# Das Geheimnis der Module

von Norbert Schlia

## Die Speicherverwaltung des 800XL

Viele Programme erhält man ausschließlich im Modul. Eigentlich ist das ja praktisch, weil die Ladezeit entfällt und die ROMs nahezu unzerstörbar sind, wenn man nicht gerade drauftritt.

Hier einige Informationen über die Technik, die das Benutzen der Module ermöglicht und wie man nahezu den gesamten, 64K langen RAM-Speicher des ATARI 800XL nutzen kann.

Gleich vorweg muß ich sagen, daß das ganze recht kompliziert wird. Einige Erfahrungen im Maschinenspracheprogrammieren sind auf jeden Fall notwendig. Aber auch für den weniger Beschlagenen kann das Folgende doch interessant sein.

## Die Verwaltungseinheiten

Der XL besitzt ein IC, die PIA, welche zusammen mit einem zweiten IC die Speicherverwaltung erledigt. Das zweite IC stellt die Memory-Management-Unit, kurz MMU, des Atari dar. Es ist ein Standardbaustein, ein PAL. Das heißt "Programmable Array Logic", die für Ataris Zwecke schon vorprogrammiert wurde. Ein solches IC besteht aus einem "Netz" von logischen Gattern. Diese werden schon bei der Herstellung je nach Wunsch des Bestellers verschaltet. So kann man bestimmte, spezielle Funktionen festlegen. Auf diese Weise ist es möglich, "maßgeschneiderte" ICs auch in kleinen Stückzahlen wirtschaftlich zu produzieren.

Die MMU kann bestimmte RAM/ROM-Blöcke austauschen. Dazu hat man den 800XL-Speicher in Blöcke aufgeteilt:

```
Block 0     $0000 - $07FF
Block 1     $0800 - $0FFF
Block 2     $1000 - $17FF

Block 30    $F000 - $F7FF
Block 31    $F800 - $FFFF
```

Wenn ein Modul eingesteckt wird, meldet es seine Existenz über zwei Leitungen, En4:H und En5:H, an. Das ":H" bedeutet, daß ein High-Pegel die entsprechende Meldung anzeigt. Man spricht dabei von "Active High", also High, wenn die Meldung "aktiv" ist. Das Gegenteil ist natürlich "Active Low" oder ":L".

Somit legt das Modul intern En5:H und/oder En4:H auf +5V, landläufig also auf High. En5 "enabled" - ermöglicht - die Umschaltung des Bereichs $A000-$BFFF. Ist das Modul 8K groß, belegt es nur diesen Bereich. Ist es 16K lang, ebenso $8000-$9FFF. Dazu setzt es auch En4 auf High.

Die MMU sorgt dabei dafür, daß das RAM in diesem Bereich ausgeblendet wird und die Daten des ROMs auf diese Adressen gelegt werden. Der Inhalt des RAM-Speichers bleibt dabei natürlich erhalten, und auch der "lebensnotwendige" Refresh wird regelmäßig durchgelassen. Dieses System nennt sich "Bank-Switching", also Speicherumschaltung.

Der sogenannte Refresh-Zyklus dient zum Erhalt der Daten im RAM. Im Atari sind ausschließlich dynamische Speicher eingebaut. Jedes der 65536 Bits wird durch einen winzigen Kondensator repräsentiert. Ist er geladen, stellt er eine "1" dar; ist er entladen, eine "0".

Jeder geladene Kondensator muß regelmäßig nachgeladen werden, damit er seine Information nicht verliert. Das erledigt die ANTIC, der Bildprozessor.

**Der Modul-Port**

| Signal | Pin | Pin | Signal |
|---|---|---|---|
| Sel4:L | 1 | A | En4:H |
| A3 | 2 | B | Masse |
| A2 | 3 | C | A4 |
| A1 | 4 | D | A5 |
| A0 | 5 | E | A6 |
| D4 | 6 | F | A7 |
| D5 | 7 | H | A8 |
| D2 | 8 | J | A9 |
| D1 | 9 | K | A12 |
| D0 | 10 | L | D3 |
| D6 | 11 | M | D7 |
| Sel5:L | 12 | N | A11 |
| +5V | 13 | P | A10 |
| En5:H | 14 | R | R:H/W:L |
| I/O5:L | 15 | S | 1.77 MHz |

Über den Port werden eine ganze Menge Informationen ausgetauscht. Will die CPU auf einen Bereich des ROMs zugreifen, wird entweder Sel4 oder Sel5 auf Low gelegt, um daß dem Modul mitzuteilen. Außerdem erhält es über Pin R die Meldung, ob es sich um eine Schreib- oder Leseoperation handelt.

Theoretisch wäre es auch möglich, nur den unteren Block zu belegen. Da aber jedes Modul dem Betriebsystem einige Informationen mitteilen muß, wie z.B. seine Startadresse, muß jedes zwangsläufig den oberen Block benutzen. An seinem Ende stehen nämlich diese Informationen:

START-ADRESSE   $BFFA/$BFFB

Der Vektor zeigt auf die Start-Adresse des Moduls.

KENN-BYTE   $BFFC

Steht hier ein Wert ungleich Null, dann "weiß" das Betriebssystem, das sich im Modul-Bereich weder ROM noch RAM befinden.

OPTIONS-BYTE   $BFFD

Von den acht Bits werden nur drei verwendet:

Bit 7:"1"  Diagnosemodul, es wird VOR der eigentlichen System-Reset-Routine initialisiert.
      "0"  Normales ROM-Modul.

Bit 2:"1"  Modul NACH dem System Reset initialisieren und starten.
      "0"  Modul nur initialisieren.

Bit 1:"1"  Vor dem Start des Moduls Diskette booten.
      "0"  Diskette nicht booten.

Alle anderen sind unbenutzt.

INIT-ADRESSE   $BFFE/$BFFF

Hier steht die Initialisierungsadresse des Moduls. Meistens zeigt sie nur auf ein RTS.

Weiteres finden Sie auch im Artikel "Epromologie, Grundwissen" aus CSM 03/1988 von P. Bee und E. Reuß.

**Der Speicherplan des ATARI 800XL**

| Adresse | | | |
|---|---|---|---|
| $0000 | RAM 32 K | | |
| $5000 $57FF | | Selftest 2 K | |
| $8000 | RAM 8 K | Cartridge (16 K) | |
| $C000 | OS-ROM 4 K | | RAM ohne OS |
| $D000 | PORTS 2 K | RAM nicht nutzbar | |
| $D800 | M.-ROM 2 K | | RAM ohne OS |
| $E000 $FFFF | OS-ROM 8 K | | |

Einige Steuersignale der MMU, die z.B. das BASIC-ROM ein und ausschalten, kann man softwaremäßig auslösen. Dazu muß man auf PORTB ($D301) der PIA bestimmte Bitkombinationen anlegen. Das Register kann auch gelesen werden, was das Setzen und Löschen von Bits mit ORA und AND erleichtert:

| Bit | Funktion |
|---|---|
| 7 | TEST:L |
| 1 | BASIC:L |
| 0 | OS-ROM:H |

Bit 2-6 werden nicht verwendet. Sie kommen erst im Zusammenhang mit RAM-Disks zu ihrem Recht. Man sollte aber, angesichts der Kompatiblität zu Erweiterungen, diese nicht verändern. Um beispielsweise das BASIC auszuschalten, kann man folgende Befehlsfolge benutzen:

```
01000 BASOFF   LDA $D301   Status laden
01010          ORA #$02    BASIC aus
01020          STA $D301   fertig
01030          RTS
```

Wieder einschalten kann man das Modul indem man in Zeile 1010 den Befehl AND #$FD gibt. Man sollte immer die Bits einODERn oder ausANDen, wie man sagt, sonst kann es sein, daß Programme nicht auf allen Systemen laufen.

Andererseits kann es auch vorkommen, daß man mit dem Programmstart einen festen Wert in das Register schreiben will. Dann sollte man $FF (ohne BASIC) bzw. $FD (mit BASIC) verwenden.

Ist Bit 0 "1", liegt im Bereich $C000-$CFFF bzw. $D800-$FFFF das Betriebssystem und in $D800-$DFFF das Mathematikroutinen-ROM.

Ist Bit 2 "0", liegt im Bereich $A000-$BFFF die BASIC-Cartridge, wie eben gezeigt.

Ist Bit 7 "0", wird der "versteckte" ROM-Bereich $D000-$D7FF nach $5000-$57FF gespiegelt. Das ist das Selbsttestprogramm des ATARI. Natürlich eine Spielerei, aber man wollte wohl die 2K-Byte Speicher nicht verschwenden.

Der Block $D000-$D7FF selbst wird als Port-Bereich genutzt. Das nennt man "Memory-Mapped-Input/Output", weil ein bestimmter Speicherbereich für alle Ein/Ausgabe- und Steueraufgaben fest abgestellt ist.

Diesen Block kann man selbstverständlich nicht abschalten, weil man ja sonst die Kontrolle über das System verlieren würde. Die 2K-Byte RAM, die hinter dem I/O-Block liegen, sind aus diesem Grund gar nicht zu erreichen.

## Das Abschalten von Blöcken

Auf das Modul im Schacht kann man softwaremäßig keinerlei Einfluß nehmen. Die Einheiten "BASIC-Modul", "OS-ROM" und "Selbsttest-ROM" jedoch kann man nach Belieben ein- und ausschalten. Dabei ist aber einiges zu beachten. Beginnen wir zunächst mit dem einfachsten:

Wer das Selbsttestprogramm von einem seiner Programme aus starten will, kann das über die Vektortabelle des ATARI machen:

JMP $E480 schaltet den Selbstest ein und startet ihn. Außerdem wird das Kaltstartflag ($244) gesetzt, damit beim nächsten RESET ein Kaltstart durchgeführt wird. Das ist wichtig, um den Selbsttest wieder verlassen zu können. Ist das ROM schon eingeblendet, kann man auch über JMP $E483 direkt hinein springen.

Es muß auch immer das Betriebsystem ein sein. Darin steht nämlich etwa von $CA57 - $CB56 der Bildschirmaufbau des Selbsttests im Bildschirmcode-Format. Er hat nicht mehr in den eigentlichen Block gepaßt. Möglicherweise werden auch Routinen aus dem Betriebssystem verwendet.

Um den Speicher hinter dem BASIC-Modul zu nutzen, kann man es leicht abschalten. Man muß nur Bit 2 von PORTB setzen. Das kann z.B. innerhalb einer Maschinenroutine geschehen, die man über USR aufgerufen hat. Dabei ist aber darauf zu achten, daß das Modul vor der Rückkehr zum BASIC wieder eingeschaltet wird!

So erhält man stolze 8K zusätzliches RAM, das zudem während des normalen BASIC-Betriebes nicht verändert wird. Das habe ich für mein Programm "RAM-Erweiterung per Software" aus CSM 1/89 genutzt. Man kann sogar ausführbare Maschinenprogramme dort ablegen, die dann allerdings über eine Hilfsroutine aufgerufen werden müssen, wenn man mit USR hinein will.

Diese Hilfsroutine muß das Modul abschalten, das Programm aufrufen und nach der Rückkkehr vom Programm das BASIC wieder ein und dann erst den Return-Befehl zum BASIC ausführen. Den Quellcode dieser kurzen Routine befindet sich im BIBO-Assemblerformat auf dieser Diskette (CALL.ASM).

Das Problem dabei ist, daß das BASIC vor dem Sprung in das Maschinenprogramm die im USR-Statement angegebenen Parameter auf dem Stack ablegt. Zunächst ein Byte, in dem die Anzahl der Parameter steht- gibt es keine, ist es Null.

Dann folgen jeweils zwei Bytes pro angegebene Größe in der Reihenfolge wie sie im Befehl standen.

Beispiel:

X=USR(ADR,A,B)

Aus dem Stack holt man mit PLA nacheinander:

2 (bedeutet "Zwei Parameter")
A (Hi), A (Lo), B (Hi), B (Lo)

Danach die Returnadresse. Die CALL-Routine korrigiert die erste Zahl, die ja eins zu hoch wäre durch den zusätzlichen Wert. Die aufgerufene Routine unter dem ROM "merkt" von der Zwischenstation nichts.

Will man dem BASIC in der im USR-Statement spezifierten Variable einen Wert übergeben, muß ihn die Maschinenroutine in der Zero-Page-Adresse $D4/$D5 bzw. 212/213 ablegen. Wird der Inhalt der Adresse nicht verändert, enthält die Variable die Startadresse der Routine.

Außerdem müssen die Maschinenprogramme vorher unter die Cartridge verschoben werden, weil sie nicht direkt dorthin geladen werden können. Dazu dient die MOVE-Routine (MOVE.ASM).

Beide können in BASIC-Strings gepackt werden und über USR(ADR(..)) aufgerufen werden. Der CALL-Routine muß dazu als ersten Parameter die Adresse der Zielroutine übergeben werden, z.B. 40960. Sie schaltet dann das BASIC ab und startet die Routine. Nach deren Ende mit RTS schaltet sie das BASIC wieder ein und kehrt zurück.

Dem USR-Statement muß also nur die Adresse der CALL-Funktion angefügt werden, um damit statt direkt zur gewünschten Adresse über CALL zu springen.

Die MOVE-Routine wird folgendermaßen aufgerufen:

X=USR(MOVE,QUELLE,ZIEL)

Die Daten werden dann von der Adresse QUELLE nach ZIEL verschoben. Schauen Sie sich doch als Demo die RAM-Erweiterung (CSM 1/89), die VBLI-Player-Steuerung (CSM 11/88) oder die Cassettendatei-Verwaltung (CSM 1/89) an. Dazu habe ich diese Technik verwendet.

Will man aus irgendeinem Grund die BASIC-Cartridge ganz abschalten, muß man außerdem Bit 2 in PORTB auch das BASIC-Flag ($3F8) setzen. Hier muß ein Wert ungleich Null stehen, sonst wird die Cartridge bei jedem RESET unweigerlich wieder eingeschaltet.

Will man das BASIC einschalten, muß man umgekehrt das Flag löschen. Danach muß man einen Kaltstart durchführen über JMP $E477. Dann wird aber auch neu gebootet. Leider ist es nicht möglich, einen Warmstart ins BASIC auszuführen, weil die Cartridge nicht korrekt initialisiert wird. Vielleicht weiß ein Leser einen geeigneten Weg?

BIBO-DOS-User haben es da noch einfacher. Herr Reuß hat für sie extra eine Möglichkeit vorgesehen, ins DOS oder zur eingesteckten Cartridge zu gelangen: JMP $715 rettet sogar das BIBO-DUP hinter das Betriebssystem, bevor es das Modul startet. Der Nachteil ist die Inkompatiblität zu anderen DOS-Versionen. Allerdings muß auch hier die Cartridge vorher schon korrekt initialisiert gewesen sein.

Ganz kompliziert wird es, wenn man mit dem Betriebssystem "spielen" will. Für die Zeit, in der das OS-ROM aus ist, müssen sämtliche Interrupts unterbunden sein. Mit dem maskierbaren ist es einfach: SEI vor und CLI nach dem Abschalten verhindert es.

Der maskierbare Interrupt, der alle 1/50 Sekunde durchgeführt wird und eventuell eingeschaltete Displaylistinterrupts, müssen schon im Auslöser, der ANTIC, gestoppt werden. Das geschieht über NMIEN ($D40E).

Hier gehört erst mal eine "0" hinein, bevor man PORTB anspricht. In diesem Moment fehlen dem 6502 nämlich die Hardwarezeiger bei $FFFA (nicht maskierbarer Interrupt), $FFFC (Reset) und $FFFE (maskierbarer Interrupt).

Die CPU würde beim nächsten Interrupt nach $0000 verzeigen, und was dann passiert, weiß der Henker. Der Reset-Vektor ist unwichtig, da beim Drücken von System-Reset das ROM sowieso wieder eingeschaltet wird. Dies als kleiner Tip, weil das bisweilen Probleme macht, wenn man Programme dahinter ablegt und sich wundert, warum sie keinen Reset überstehen!

$40 in NMIEN schaltet den Interrupt wieder ein. Man kann es auch, nachdem das OS-ROM wieder eingeblendet ist, über JSR $E46B anschalten. Falls VBLI im Spiel waren, ist $C0 der richtige Wert.

Ganz nebenbei muß der Programmierer, will er ein Chaos auf dem Bildschirm vermeiden, auch einen vernünftigen Zeichensatz bereitstellen- vorausgesetzt, er will das Betriebssystem längere Zeit aus lassen.

Mit diesen Tricks ist es möglich, insgesamt 62K RAM zu mobilisieren und auch zu nutzen. Ich hoffe, damit einige brauchbare Tips gegeben zu haben. Ich wünsche allen ATARI-Usern viel Spaß mit ihren Geräten!